提高软件可修改性的若干方法

林志斌 （交通部广州信息技术研究所 510075）

摘要：本文结合笔者开发 MIS 软件的经验，提出了提高软件可修改性的若干方法：使用数据库、修改输入域、信息屏蔽、提高模块独立性、面向对象程序设计、为修改留下余地。

关键词：软件 可修改性 数据库 信息屏蔽 独立性 面向对象

软件的可维护性包括可理解性，可测试性，可修改性。提高软件的可修改性能提高软件的可维护性。

下面具体谈谈提高软件可修改性的几种方法。

一、使用数据库

尽管应用软件可执行数据完整性检查，但若让数据库来处理这个问题会更简单，出错的可能性也较小。如若特定课程具有先修课，并限制人数或其他别的限制条件，那么如果应用软件负责检查这些限制，则安排学生修课的每个应用程序都必须包含这些条件检查的代码，并且每次条件变化时，则所有的应用程序都必须修改。

相反，数据库则不同，它把条件放在数据库中，因而可以容易地更改数据库以反映限制条件的改变。也就是说把数据修改的任务留给了数据库本身。

现在流行的通用下拉式菜单技术就是把菜单存放在数据库中。

首先定义一个数据库，存放可变的菜单信息，它的字段结构描述如下：

字段名	类型	宽度	用途
BH	字符型	5	存放菜单项目编号，第一位表示第一层菜单的项目编号；第二位表示第二层菜单的项目编号；第五位表示第五层菜单的项目编号。
XX	字符型	20	存放菜单项目信息
MK	字符型	11	存放该项目执行的模块名，如果为空，则继续调用下级菜单，RETURN 返回上级菜单。

数据库的内容可根据不同的应用自行设置。例如

BH	XX	MK
1	发票处理	
2	提油单处理	
3	验收单处理	
4	退出	QUIT
11	发票录入	FPLR
12	发票修改	FPXG
13	发票查询	FPCX
14	返回	RETURN
.....		

通用菜单管理程序读数据库，取菜单项目编号、菜单项目信息、要执行模块名称。由于菜单信息包含在数据库中，所以修改起来很方便。

二、修改输入域

软件修改时往往要带来这样那样的错误或者副作用，即通常所说的波及影响（ripple effect），所以应尽量避免修改软件本身。而输入域的修改是容易的。

假设程序中用到变量 row，并要多次对其赋初值 5。

实现方法有三种：

1．用赋值语句 row = 5

重新置初值时再用该语句赋值。

如果 ROW 的初值要改变，如要改为 6，则程序中所有的 row = 5 语句都要改成 row = 6。

这样很繁，而且容易出错。

2．把初值先赋给一个变量，然后再把该变量赋给 row。

例：origin = 5

row = origin

重新置初值时用 row = origin 语句赋值。

row 的初值要改变时，例如要改为 6，仅将 origin = 5 改为 origin = 6 就可以了。

可修改性大大提高。

3．row 的初值作为命令行参数，传递给变量 origin，

用 row = origin 语句给 row 置初值。设文件名为 play，以 FoxBASE 语言为例，执行时用 DO play WITH 5 即可。

初值被读入该软件，而不是嵌入程序。

这样一来，不用改程序本身，只要改命令行参数则可，方便、灵活。

上述三种方法中，可修改性从低到高。第一种方法不足取，第三种方法可修改性最优。第二种方法在实践中也广泛采用，仍不失为一种好方法。例如：

假设程序中要多次用 set colo to 语句设置颜色，开始选用红色 red，所以用 set colo to r 语句，后觉得红色太刺眼，想改为蓝色 blue，于是程序中所有 set colo to r 语句都要改成 set colo to b，很不方便。如采用如下方法，则修改起来很方便：

c = "b", set colo to &c。

即先将颜色赋给变量 c，然后用宏替换 &c 代表颜色。要改变颜色时只要把新的颜色赋给变量 c 即可。

例如，改蓝色后觉蓝色不是合适的前境色，想改为绿色 green，于是改 c = 'b' 为 c = 'g'。

使用宏替换能大大提高可修改性。

当然，如果将颜色作为命令行参数，则修改会更方便。

三、信息屏蔽

大型软件系统一般是多个版本的，在整个生命期中往往要经受多次修改，所以在分解模块时就应采取措施使将来修改造成的影响尽可能局限在一个或少数几个模块的内部，为了达到这个目的，Parnas 提出了信息屏蔽(Information Hiding)的原则。

根据信息屏蔽原则，概要设计可以这样进行：

1. 列出可能发生变化的因素，把它们放在一起。

2. 划分模块时将一些可能发生变化的因素隐含在某个模块内部，使其他模块与此因素无关。对于这样构造的软件系统，将来这些因素发生变化需作修改维护时，只要改一个模块就够了，其他模块可不受影响。也就是说，信息屏蔽技术将某个因素隔离在一个模块内部，这个因素的变化不致于传播到所在模块的边界之外。由于修改极易引起错误，修改的影响范围越小，则修改引起错误的可能性越小。

例如，可以把依赖于具体运行环境和功能需求的部分集中在少数几个模块内，一旦环境和功能需求发生变化，就可以用其他模块代替，通过一些局部性的修改，在仍然保持整体稳定性的前提下，适应新的环境和需求。

四、提高模块独立性

模块化能够提高软件的可修改性。

开发具有独立功能而且和其他模块间没有过多的相互作用的模块，就可以做到模块独立。

独立的模块容易维护，修改程序需要的工作量较小，错误传播范围小。

模块的独立程度可由两个定性标准度量，即耦合和聚合，即块间联系和块内联系。

耦合衡量不同模块彼此间互相依赖的紧密程度。

块间联系越小，模块的独立性越高。在设计模块结构图时，应尽量通过过程调用语句实现模块调用。模块调用时应传送数据型参数，传送的数据应尽可能少，同时应尽量保持模块界面的清晰性，以减少模块间的块间联系，提高模块的独立性。聚合衡量一个模块内部各个元素彼此结合的紧密程度。

块内联系按从强到弱可分成六种不同类型，即功能性、顺序性、通信性、瞬时性、逻辑性、偶然性。功能性、顺序性、通信性三种块内联系是由于模块中的成分共用数据而引起的，因而联系比较强。而瞬时性、逻辑性、偶然性三种块内联系不是由成分间共同信息引起的，因而联系比较弱。

改进软件结构可提高模块独立性。设计出软件的初步结构以后应该审查分析这个结构，通过模块分解或合并，力求降低耦合，提高聚合。例如多个模块公有的一个子功能可以独立成一个模块，由这些模块调用，有时可以通过分解或合并模块以减少控制信息的传递及对全局数据的引用，并且降低接口的复杂程度。

五、面向对象技术

九十年代面向对象模型的实现使程序员和最终用户都受益非浅。对象的一个重要特点是封闭性，它是数据和过程紧密结合在一起的整体，具有很强的独立性，是实现模块的理想机制。从外面只能看到对象的外部特性，即能够接受那些信息，具有哪些处理能力，而对象的内部特性，即实现处理能力的算法和保存内部状态的私有数据对外是不可见的。对私有数据不能执行该对象成员函数之外任何其他操作，对象的这种性质称为信息屏蔽。程序员只需知道类的外部说明就能使用它们，完全不必

知道私有数据的说明，也不必知道函数的实现算法，修改内部数据结构和函数的算法完全不影响其他程序成分的实现。信息屏蔽大大提高了程序的可修改性。

面向对象程序设计语言的继承性，使得用户在修改和扩充程序时不必修改原有的程序代码，只需增加新的代码，因而不必知道原有程序是怎样实现的，极大地减少了维护工作量。

六、为修改留下余地

软件在设计、编程和测试时应注意到将来不可避免的修改，必须时时处处为软件维护着想，为修改留下余地。

如程序在编制时，应尽量注意可用资源（主存、磁盘等）的节约使用，以便为修改留有余地。

又如在一个文件里，页码、图号和表格号都按对应的章节编制，即 1－1，1－2，……，2－1，2－2，…等等，以便插入或删除，而不会引起全面的不必要的重新编制文件号码。又如一个程序不要写成一大整块，否则不适于分段进入，不能为修改留有余地。

参考文献

[1] 郑人杰 《高级程序员水平考试指导》，清华大学出版社．1991：190

[2] 张海潘 《软件工程导论》，清华大学出版社．1995：213

[3] （美）B．W．博姆等 《软件质量的鉴定》，科学普及出版社．1985：51

（来稿时间：1996 年 6 月）